An Introduction to Structured Query Language

Grant Weddell

Cheriton School of Computer Science
University of Waterloo

CS 348
Introduction to Database Management
Winter 2017

Outline

Structured Query Language

Structured Query Language (SQL) is made up of two sub-languages:

Structured Query Language

Structured Query Language (SQL) is made up of two sub-languages:

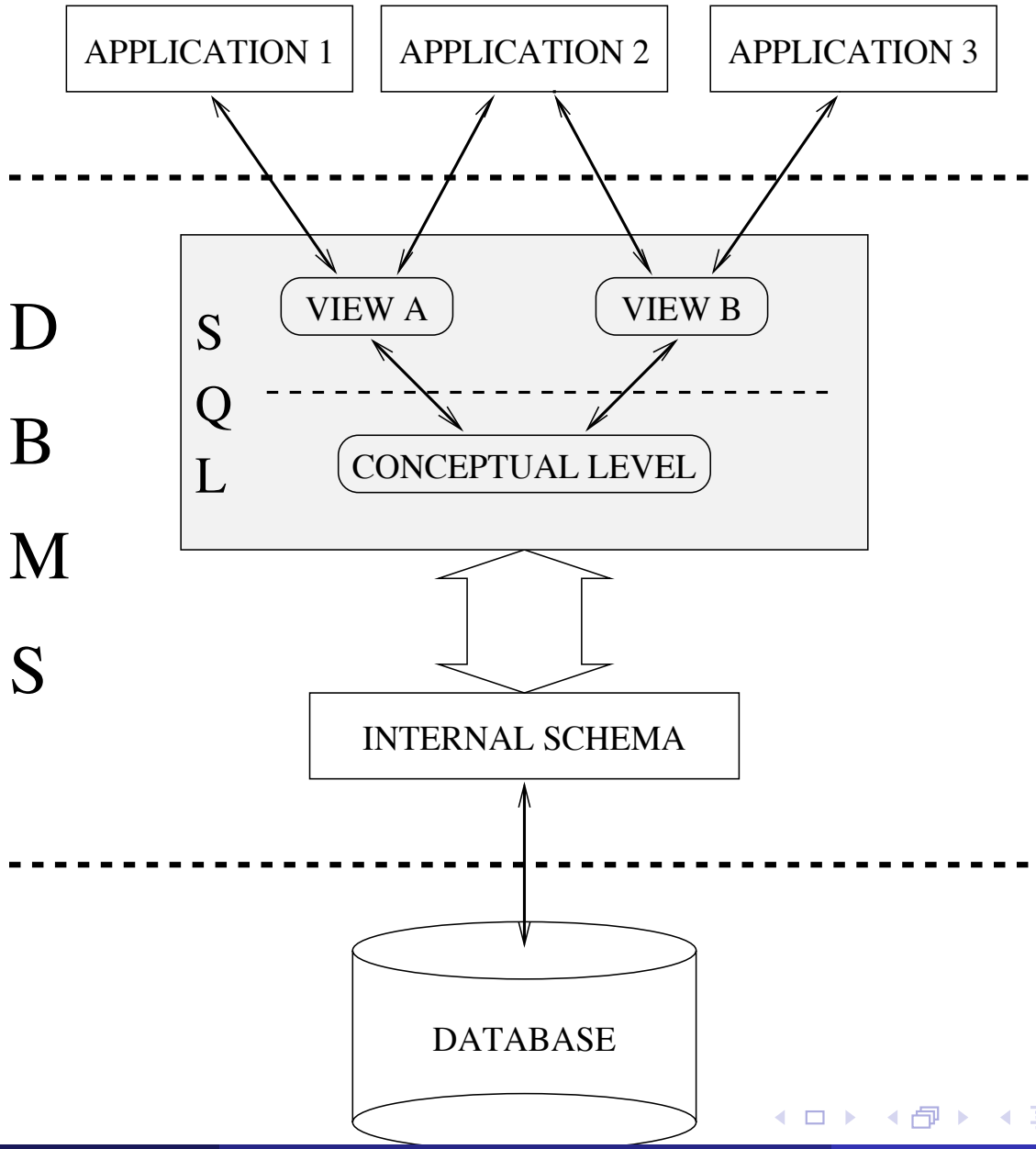
- SQL Data Manipulation Language (DML)
  - `SELECT` statements perform queries
  - `INSERT`, `UPDATE`, `DELETE` statements modify the instance of a table

Structured Query Language

Structured Query Language (SQL) is made up of two sub-languages:

- SQL Data Manipulation Language (DML)
  - `SELECT` statements perform queries
  - `INSERT`, `UPDATE`, `DELETE` statements modify the instance of a table
- SQL Data Definition Language (DDL)
  - `CREATE`, `DROP` statements modify the database schema
  - `GRANT`, `REVOKE` statements enforce the security model

The SQL Standard



SQL DML: Queries

```
select LastName,HireDate
from Employee
where Salary > 100000
```

Find the last names and hire dates of employees who make more than $100000.

Note

SQL is declarative (non-navigational)

Multisets

- Relational model: relations are sets
- SQL standard: tables are multisets (a.k.a. bags)
  - Duplicate tuples may be stored.
  - SQL queries may result in duplicates even if none of the input tables themselves contain duplicates.
  - `Select distinct` is used to eliminate duplicates from the result of a query.

```
select distinct LastName,HireDate
from Employee
where Salary > 100000
```

SQL Query Involving Several Relations

```
select P.ProjNo, E.LastName
from Employee E, Project P
where P.RespEmp = E.EmpNo
  and P.DeptNo = 'E21'
```

For each project for which department E21 is responsible, find the name of the employee in charge of that project.

The SQL Basic Query Block

```
select attribute-expression-list
from relation-list
[where condition]
```

Note

The result of such a query is a relation which has one attribute for each element of the query's attribute-expression-list.

The SQL "Where" Clause

Conditions may include

- arithmetic operators +, -, *, /
- comparisons =, <>, <, <=, >, >=
- logical connectives **and**, **or** and **not**

```
select E.LastName
from Employee E,
       Department D,
       Employee Emgr
where E.WorkDept = D.DeptNo
  and D.MgrNo = Emgr.EmpNo
  and E.Salary > Emgr.Salary
```

List the last names of employees who make more than their manager.

The SQL "Select" Clause

- Return the difference between each employee's actual salary and a base salary of $40000

```
select E.EmpNo, E.Salary - 40000 as SalaryDiff
from Employee E
```

The SQL "Select" Clause

- Return the difference between each employee's actual salary and a base salary of $40000

```
select E.EmpNo, E.Salary - 40000 as SalaryDiff
from Employee E
```

- As above, but report zero if the actual salary is less than the base salary

```
select E.EmpNo,
       case when E.Salary < 40000 then 0
            else E.Salary - 40000 end
from Employee E
```

SQL DML: Insertion & Deletion

```
insert into Employee
values ('000350',
        'Sheldon', 'Q',
        'Jetstream',
        'A00',
        01/10/2000,
        25000.00);
```

Insert a single tuple into the Employee relation.

```
delete from Employee;
```

Delete all employees from the Employee table.

```
delete from Employee
where WorkDept = 'A00';
```

Delete all employees in department A00 from the Employee table.

SQL DML: Update

```
update Employee
set Salary = Salary * 1.05;
```

Increase the salary of every employee by five percent.

```
update Employee
set WorkDept = 'E01'
where WorkDept = 'E21';
```

Move all employees in department E21 into department E01.

Set Operations

- SQL defines `UNION`,`INTERSECT` and `EXCEPT` operations (`EXCEPT` is set difference)

```
select empno
from employee
except
select mgrno
from department
```

Set Operations

- SQL defines `UNION`,`INTERSECT` and `EXCEPT` operations (`EXCEPT` is set difference)

```
select empno
from employee
except
select mgrno
from department
```

- These operations result in sets
  - Q_1 `UNION` Q_2 includes any tuple that is found (at least once) in Q_1 or in Q_2
  - Q_1 `INTERSECT` Q_2 includes any tuple that is found (at least once) in both Q_1 and Q_2
  - Q_1 `EXCEPT` Q_2 includes any tuple that is found (at least once) in Q_1 and is not found Q_2

Multiset Operations

- SQL provides a multiset version of each of the set operations: `UNION ALL`, `INTERSECT ALL`, `EXCEPT ALL`

Multiset Operations

- SQL provides a multiset version of each of the set operations: `UNION ALL`, `INTERSECT ALL`, `EXCEPT ALL`
- suppose Q_1 includes n_1 copies of some tuple t, and Q_2 includes n_2 copies of the same tuple t.
  - Q_1 `UNION ALL` Q_2 will include $n_1 + n_2$ copies of t
  - Q_1 `INTERSECT ALL` Q_2 will include $\min(n_1, n_2)$ copies of t
  - Q_1 `EXCEPT ALL` Q_2 will include $\max(n_1 - n_2, 0)$ copies of t

NULL values

- the value NULL can be assigned to an attribute to indicate unknown or missing data

NULL values

- the value NULL can be assigned to an attribute to indicate unknown or missing data
- NULLs are a necessary evil - lots of NULLs in a database instance suggests poor schema design

NULL values

- the value NULL can be assigned to an attribute to indicate unknown or missing data
- NULLs are a necessary evil - lots of NULLs in a database instance suggests poor schema design
- NULLs can be prohibited for certain attributes by schema constraints, e.g., `NOT NULL`, `PRIMARY KEY`

NULL values

- the value NULL can be assigned to an attribute to indicate unknown or missing data
- NULLs are a necessary evil - lots of NULLs in a database instance suggests poor schema design
- NULLs can be prohibited for certain attributes by schema constraints, e.g., `NOT NULL, PRIMARY KEY`
- predicates and expressions that involve attributes that may be NULL may evaluate to NULL
  - $x + y$ evaluates to NULL if either x or y is NULL
  - $x > y$ evaluates to NULL if either x or y is NULL
  - how to test for NULL? Use `is NULL` or `is not NULL`

Note

SQL uses a three-valued logic: TRUE, FALSE, NULL

Logical Expressions in SQL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Logical Expressions in SQL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Logical Expressions in SQL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

NULL and the SQL Where Clause

- The query:

```
select *
from employee
where hiredate <> '05/05/1947'
```

  will *not* return information about employees whose hiredate is NULL.

Note

The condition in a **where** *clause filters out any tuples for which the condition evaluates to FALSE or to NULL.*

Subqueries

- These two queries are equivalent.

```
select deptno, deptname
from department d, employee e
where d.mgrno = e.empno and e.salary > 50000
```

```
select deptno, deptname
from department
where mgrno in
   ( select empno
     from employee
     where salary > 50000 )
```

Subquery Constructs in SQL

- SQL supports the use of the following predicates in the **where** clause. A is an attribute, Q is a query, op is one of $>, <, <>, =, <=, >=$.
  - A IN (Q)
  - A NOT IN (Q)
  - A op SOME (Q)
  - A op ALL (Q)
  - EXISTS (Q)
  - NOT EXISTS (Q)
- For the first four forms, the result of Q must have a single attribute.

Another Subquery Example

- Find the name(s) and number(s) of the employee(s) with the highest salary.

```
select empno, lastname
from employee
where salary >= all
        ( select salary
          from employee )
```

Another Subquery Example

- Find the name(s) and number(s) of the employee(s) with the highest salary.

```
select empno, lastname
from employee
where salary >= all
        ( select salary
          from employee )
```

Note

Is this query correct if the schema allows the salary attribute to contain NULLs?

Correlated Subqueries

- This query also returns the employee(s) with the largest salary:

```
select empno, lastname
from employee E1
where salary is not null and not exists
        ( select *
          from employee E2
          where E2.salary > E1.salary)
```

- This query contains a *correlated* subquery - the subquery refers to an attribute (E1.salary) from the outer query.

Scalar Subqueries

- Subquery that returns an atomic value (one row / one column)

Scalar Subqueries

- Subquery that returns an atomic value (one row / one column)
- in the **where** clause:

```
select empno, lastname
from employee
where salary >
          (select salary
           from employee e2
           where e2.empno = '000190')
```

Scalar Subqueries

- Subquery that returns an atomic value (one row / one column)
- in the **where** clause:

```
select empno, lastname
from employee
where salary >
        (select salary
         from employee e2
         where e2.empno = '000190')
```

- in the **select** clause:

```
select projno,
       (select deptname
        from department d
        where e.workdept = d.deptno)
from project p, employee e
where p.respemp = e.empno
```

Table Expressions

- in the **from** clause:

```
select projno, projname
from project p,
   (select mgrno
    from department, employee
    where mgrno = empno and salary > 100000) as m
where respemp = mgrno
```

Table Expressions

- in the **from** clause:

```
select projno, projname
from project p,
   (select mgrno
    from department, employee
    where mgrno = empno and salary > 100000) as m
where respemp = mgrno
```

- in a **with** clause:

```
with Mgrs(empno) as
   (select mgrno
    from department, employee
    where mgrno = empno and salary > 100000)
select projno, projname
from project, Mgrs
where respemp = empno
```

Outer Joins

- List the manager of each department. Include in the result departments that have no manager.

```
select deptno, deptname, lastname
from department d left outer join employee e
          on d.mgrno = e.empno
where deptno like 'D%'
```

Outer Joins

- List the manager of each department. Include in the result departments that have no manager.

```
select deptno, deptname, lastname
from department d left outer join employee e
        on d.mgrno = e.empno
where deptno like 'D%'
```

Note

SQL supports left, right, and full outer joins.

Ordering Results

- No particular ordering on the rows of a table can be assumed when queries are written. (This is important!)
- No particular ordering of rows of an intermediate result in the query can be assumed either.
- However, it is possible to order the final result of a query, using the **order by** clause.

```
select distinct e.empno, emstdate, firstnme, lastname
from employee e, emp_act a
where e.empno = a.empno and a.projno = 'PL2100'
order by emstdate
```

Grouping and Aggregation: An Example

- For each department, list the number of employees it has and their combined salary.

```
select deptno, deptname, sum(salary) as totalsalary,
        count(*) as employees
from department d, employee e
where e.workdept = d.deptno
group by deptno, deptname
```

Grouping and Aggregation: Operational Semantics

- The result of a query involving grouping and aggregation can be determined as follows:
  1. form the cross product of the relations in the **from** clause
  2. eliminate tuples that do not satisy the condition in the **where** clause
  3. form the remaining tuples into groups, where all of the tuples in a group match on all of the grouping attributes
  4. generate one tuple per group. Each tuple has one attribute per expression in the **select** clause.
- aggregation functions are evaluated separately for each group

Grouping and Aggregation Example

- Apply **where**

DEPTNO	DEPTNAME	SALARY
A00	SPIFFY COMPUTER SERVICE DIV.	52750.00
A00	SPIFFY COMPUTER SERVICE DIV.	46500.00
B01	PLANNING	41250.00
C01	INFORMATION CENTER	38250.00
D21	ADMINISTRATION SYSTEMS	36170.00
D21	ADMINISTRATION SYSTEMS	22180.00
D21	ADMINISTRATION SYSTEMS	19180.00
D21	ADMINISTRATION SYSTEMS	17250.00
D21	ADMINISTRATION SYSTEMS	27380.00
E01	SUPPORT SERVICES	40175.00
E11	OPERATIONS	29750.00
E11	OPERATIONS	26250.00
E11	OPERATIONS	17750.00
E11	OPERATIONS	15900.00
E21	SOFTWARE SUPPORT	26150.00

Grouping and Aggregation Example (cont'd)

- Apply **where**, then **group by**

DEPTNO	DEPTNAME	SALARY
A00	SPIFFY COMPUTER SERVICE DIV.	52750.00
A00	SPIFFY COMPUTER SERVICE DIV.	46500.00
B01	PLANNING	41250.00
C01	INFORMATION CENTER	38250.00
D21	ADMINISTRATION SYSTEMS	36170.00
D21	ADMINISTRATION SYSTEMS	22180.00
D21	ADMINISTRATION SYSTEMS	19180.00
D21	ADMINISTRATION SYSTEMS	17250.00
D21	ADMINISTRATION SYSTEMS	27380.00
E01	SUPPORT SERVICES	40175.00
E11	OPERATIONS	29750.00
E11	OPERATIONS	26250.00
E11	OPERATIONS	17750.00
E11	OPERATIONS	15900.00
E21	SOFTWARE SUPPORT	26150.00

Grouping and Aggregation Example (cont'd)

- Finally project and aggregate

DEPTNO	DEPTNAME	TOTALSALARY	EMPLOYEES
A00	SPIFFY COMPUTER SERVICE DIV.	99250.00	2
B01	PLANNING	41250.00	1
C01	INFORMATION CENTER	38250.00	1
D21	ADMINISTRATION SYSTEMS	122160.00	5
E01	SUPPORT SERVICES	40175.00	1
E11	OPERATIONS	89650.00	4
E21	SOFTWARE SUPPORT	26150.00	1

Aggregation Functions in SQL

- count(*): number of tuples in the group
- count(E): number of tuples for which E (an expression that may involve non-grouping attributes) is non-NULL
- count(distinct E): number of distinct non-NULL E values
- sum(E): sum of non-NULL E values
- sum(distinct E): sum of distinct non-NULL E values
- avg(E): average of non-NULL E values
- avg(distinct E): average of distinct non-NULL E values
- min(E): minimum of non-NULL E values
- max(E): maximum of non-NULL E values

The Having Clause

- List the average salary for each large department.

```
select deptno, deptname, avg(salary) as MeanSalary
from department d, employee e
where e.workdept = d.deptno
group by deptno, deptname
having count(*) >= 4
```

Note

The where clause filters tuples before they are grouped, the having clause filters groups.

Grouping and Aggregation: Operational Semantics

- The result of a query involving grouping and aggregation can be determined as follows:
  1. form the cross product of the relations in the **from** clause
  2. eliminate tuples that do not satisy the condition in the **where** clause
  3. form the remaining tuples into groups, where all of the tuples in a group match on all of the grouping attributes
  4. eliminate any groups of tuples for which the **having** clause is not satisfied
  5. generate one tuple per group. Each tuple has one attribute per expression in the **select** clause.
- aggregation functions are evaluated separately for each group

Grouping and Aggregation with Having

- Apply **where**, then **group by**

DEPTNO	DEPTNAME	SALARY
A00	SPIFFY COMPUTER SERVICE DIV.	52750.00
A00	SPIFFY COMPUTER SERVICE DIV.	46500.00
B01	PLANNING	41250.00
C01	INFORMATION CENTER	38250.00
D21	ADMINISTRATION SYSTEMS	36170.00
D21	ADMINISTRATION SYSTEMS	22180.00
D21	ADMINISTRATION SYSTEMS	19180.00
D21	ADMINISTRATION SYSTEMS	17250.00
D21	ADMINISTRATION SYSTEMS	27380.00
E01	SUPPORT SERVICES	40175.00
E21	SOFTWARE SUPPORT	26150.00
E11	OPERATIONS	29750.00
E11	OPERATIONS	26250.00
E11	OPERATIONS	17750.00
E11	OPERATIONS	15900.00

Grouping and Aggregation with Having (cont'd)

- After grouping, apply **having**

DEPTNO	DEPTNAME	SALARY
D21	ADMINISTRATION SYSTEMS	36170.00
D21	ADMINISTRATION SYSTEMS	22180.00
D21	ADMINISTRATION SYSTEMS	19180.00
D21	ADMINISTRATION SYSTEMS	17250.00
D21	ADMINISTRATION SYSTEMS	27380.00
E11	OPERATIONS	29750.00
E11	OPERATIONS	26250.00
E11	OPERATIONS	17750.00
E11	OPERATIONS	15900.00

Grouping and Aggregation with Having (cont'd)

- After grouping, apply **having**

DEPTNO	DEPTNAME	SALARY
D21	ADMINISTRATION SYSTEMS	36170.00
D21	ADMINISTRATION SYSTEMS	22180.00
D21	ADMINISTRATION SYSTEMS	19180.00
D21	ADMINISTRATION SYSTEMS	17250.00
D21	ADMINISTRATION SYSTEMS	27380.00
E11	OPERATIONS	29750.00
E11	OPERATIONS	26250.00
E11	OPERATIONS	17750.00
E11	OPERATIONS	15900.00

- Finally project and aggregate

DEPTNO	DEPTNAME	MEANSALARY
D21	ADMINISTRATION SYSTEMS	24432.00
E11	OPERATIONS	22412.50

Selecting Non-Grouping Attributes

```
db2 => select deptno, deptname, sum(salary) \
db2 (cont.) => from department d, employee e \
db2 (cont.) => where e.workdept = d.deptno \
db2 (cont.) => group by deptno
SQL0119N  An expression starting with "DEPTNAME"
specified in a SELECT clause, HAVING clause, or
ORDER BY clause is not specified in the GROUP BY
clause or it is in a SELECT clause, HAVING clause,
or ORDER BY clause with a column function and no
GROUP BY clause is specified.  SQLSTATE=42803
```

Note

Non-grouping attributes may appear in the **select** *clause only in aggregate expressions. (Why?)*

SQL DDL: Tables

```
create table Employee (
    EmpNo char(6),
    FirstName varchar(12),
    MidInit char(1),
    LastName varchar(15),
    WorkDept char(3),
    HireDate date
)

alter table Employee
  add column Salary decimal(9,2)

drop table Employee
```

SQL DDL: Data Types

Some of the attribute domains defined in SQL:

- INTEGER
- DECIMAL(p,q): p-digit numbers, with q digits right of decimal
- FLOAT(p): p-bit floating point numbers
- CHAR(n): fixed length character string, length n
- VARCHAR(n): variable length character string, max. length n
- DATE: describes a year, month, day
- TIME: describes an hour, minute, second
- TIMESTAMP: describes and date and a time on that date
- YEAR/MONTH INTERVAL: time interval
- DAY/TIME INTERVAL: time interval
- ...

Integrity Constraints in SQL

Most commonly-used SQL schema constraints:

- Not Null
- Primary Key
- Unique
- Foreign Key
- Column or Tuple Check

Integrity Constraints in SQL

Most commonly-used SQL schema constraints:

- Not Null
- Primary Key
- Unique
- Foreign Key
- Column or Tuple Check

Note

Recent SQL standards also allows more powerful integrity constraints. However, they are not supported by all commercial DBMSs.

SQL DDL: Integrity Constraints

```
create table Employee (
    EmpNo char(6) not null primary key,
    FirstName varchar(12) not null,
    MidInit char(1),
    LastName varchar(15) not null,
    WorkDept char(3) not null references Department
                                    on delete cascade,
    HireDate date,
    Salary decimal(9,2) check (Salary >= 10000),
    constraint unique_name_dept
        unique (FirstName, LastName, WorkDept)
)

alter table Employee
  add column StartDate date
  add constraint hire_before_start
        check (HireDate <= StartDate);
```

Another SQL Constraint Example

```
create table registeredin (
  coursenum char(5) not null,
  term char(3) not null,
  id char(8) not null references student
                          on delete no action,
  sectionnum char(2) not null,
  mark integer,
  constraint mark_check check (
      mark >= 0 and mark <= 100 ),
  primary key (coursenum, term, id),
  foreign key (coursenum, sectionnum, term)
    references section
)
```

More Powerful SQL Integrity Constraints

```
create assertion balanced_budget check (
not exists (
    select deptno
    from department d
    where budget <
        (select sum(salary)
         from employee
         where workdept = d.deptno)))
```

Note

General assertions are not supported by current versions of DB2.

Triggers

Definition

A *trigger* is a procedure executed by the database in response to a change to the database instance.

Triggers

Definition

A *trigger* is a procedure executed by the database in response to a change to the database instance.

Basic components of a trigger description:

- **Event:** Type of change that should cause trigger to fire
- **Condition:** Test performed by trigger to determine whether further action is needed
- **Action:** Procedure executed if condition is met

SQL DDL: Trigger Example

```
create trigger log_addr
    after update of addr, phone on person
    referencing OLD as o NEW as n
    for each row
    mode DB2SQL  /* DB2-specific syntax */
    when (o.status = 'VIP' or n.status = 'VIP')
        insert into VIPaddrhist(pid, oldaddr, oldphone,
            newaddr, newphone, user, modtime)
        values (o.pid, o.addr, o.phone,
            n.addr, n.phone, user, current timestamp)
```